SAND2009-0203P

HETEROGENEOUS ENSEMBLE CLASSIFICATION

SEAN A. GILPIN* AND DANIEL M. DUNLAVY†

Abstract. The problem of multi-class classification is explored using heterogeneous ensemble classifiers. Heterogeneous ensembles classifiers are defined as ensembles, or sets, of classifier models created using more than one type of classification algorithm. For example, the outputs of decision tree classifiers could be combined with the outputs of support vector machines (SVM) to create a heterogeneous ensemble. We explore how, when, and why heterogeneous ensembles should be used over other classification methods. Specifically we look into the use of bagging and different fusion methods for heterogeneous and homogeneous ensembles. We also introduce the HEMLOCK framework, a software tool for creating and testing heterogeneous ensembles.

1. Introduction. The problem of data classification, or data labeling, arises in a wide variety of applications. Examples include detecting spam e-mail messages based on the content of the messages (document classification), labeling cells and tumors as malignant or benign based on the context of MRI scan data (image classification), and identification of individuals based on fingerprints, facial features, and iris patterns (biometric identification). In all of these examples, the goal is to predict a discrete label (e.g., "spam" versus "not spam") for a particular data instance (e.g., a particular e-mail message) based on the attributes of that instance.

More formally, classification is the task of learning a function, f, that maps a set of data instance attributes, $\mathbf{x} = \langle a_1(\mathbf{x}), \ldots, a_m(\mathbf{x}) \rangle$, to one of several predefined class labels, $\mathcal{Y} = \{y_1, \ldots, y_k\}$. When a data instance can be deduced easily from the context, attribute j of that instances will be denoted simply as a_j. The function f is often called a classifier, classifier model, or hypothesis. The set of data instances used to learn, or train, a classifier model is called the training set and is denoted $\mathcal{D}_{tr} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where n is the number of instances, $\mathbf{x} \in \mathbb{R}^m$ is a vector of attributes, or features, for data instance i, and y_i is the label for data instance i. In order to validate the models learned, it is common practice to select some of the training data to be used in testing the resulting classifier models. This testing, or validation data, is denoted $\mathcal{D}_{te}$ and is not used in training the classifier model. Throughout this paper, we assume that all labels given for the training and testing data are correct—i.e., there are no mislabeled instances.

Recent results in solving classification problems indicate that the use of ensembles, or sets of classifier models, often leads to improved performance over using single classifier models [3, 4, 5, 24]. Much of the previous work on ensembles of classifier models (see e.g., [7]) has focused on *homogeneous ensemble classifiers*—i.e., collections of classifier models of a single type. In this work, we focus on *heterogeneous ensemble classifiers*, where the collection of classifiers are not of the same type. Note that such classifier models are also referred to as hybrid ensemble classifiers. Our goal is to find when and how the use of heterogeneous ensembles can be advantageous.

The motivation for our current work stems from previous work in classifying text documents [4]. The problems in that domain sometimes involve two classes (e.g., "spam" versus "not spam" in the e-mail classification problem), but more generally involve more than two classes (e.g., mapping scientific articles to appropriate journals for publication). Thus, we focus on the general problem of multi-class classification in this paper (i.e., $k >= 2$). We are also interested in incorporating data with missing

*Computer Science, San Jose State University, sgilpin80@gmail.com
†Computer Science and Informatics, Sandia National Laboratories, dmdunla@sandia.gov

attributes or with both continuous and discrete attributes into our models, as such data often arises in text document classification problems (e.g., documents do not contain all terms [i.e., features] and documents can contain both continuous features [via vector space models] and discrete features [dates, publication names, etc.]).

As part of this work, we have created a software framework called HEMLOCK (Heterogeneous Ensemble Machine Learning Open Classification Kit) for creating and evaluating heterogeneous ensemble classifiers. Although the methods described in this paper for classifier models, ensemble creation, and classifier validation/performance are applicable to the problem of classification in general, the majority of the focus is on those methods currently available in HEMLOCK.

2. Methods. In this section, we describe the methods used in the HEMLOCK software package to generate classifier models. Currently, HEMLOCK interfaces a software library called WEKA [25] for all of its classification methods. These methods formulate models that include mathematical descriptions of decision boundaries—i.e., hyperplanes, piecewise hyperplanes, or nonlinear manifolds that partition the feature vector space induced from a given training set of data. Combined with decision rules particular to each method, these decision boundaries are used to determine which class labels are associated with different areas of the feature space. Note that some methods generate explicit representations of the decision boundaries via parameters of some explicit function (e.g., support vector machines), whereas others generate implicit boundaries (e.g., nearest neighbor classifiers). Appendix A has information about the methods used from WEKA. The rest of this section focuses on the ensemble methods implemented in HEMLOCK. Throughout this section, "data" refers to "training data" unless otherwise indicated.

2.1. Ensemble Classifiers. Ensemble classifiers are a type of meta-model that use a set of base classifiers as input to a combination function. The combination function is intended to make the best use of the information provided from the base classifiers in order to make class label predictions as accurately as possible. These ensembles are homogeneous, referring to the fact that all of the base classifiers are of a single type (e.g., decision trees), differing by model parameters, the data used for training, or a combination of the two.

Ensemble classifiers have been found to be generally more accurate than non-ensemble classifiers. Following are different situations in which using an ensemble classifier model over a single classifier model have led to improved classifier performance in practice [11].

- Base classifiers may not be able to model the true class decision boundaries exactly. For example, a linear model can never exactly represent a quadratic decision boundary. However an ensemble with linear models as base classifiers will in general lead to more flexible decision boundaries than those of the simple, underlying linear models.
- A lack of data can lead to many good estimations of the true class boundaries. Instead of choosing one, an ensemble can use all of them as base classifiers to eliminate the chance of picking the worst classifier.
- Globally optimal searches of the classifier function space are not computationally feasible for large sets of data or data instances with large numbers of features. Most classification algorithms are therefore limited to searches that lead to locally optimal parametrization. In this case, a combination of models may better approximate globally optimal estimation of true decision

boundaries by employing base classifiers that search different regions of the global classifier parameter space.
- Noise in the training data can be addressed by combining models that are trained using data sampled from the entire training data set. Combinations of models trained is such a way often reduce overfitting the data as well.

There are two general types of ensemble combination functions: fusion and selection. In fusion functions, the output from each of the base classifiers includes a weight on every prediction made by the ensemble. A typical fusion function is the sum of the weighted predictions. Selection functions, on the other hand, allow the use of outputs from one or more of the base classifiers, not necessarily making use of the outputs of all of the predictions. Moreover, the base classifier output used in the selection function typically depends on the characteristics of the instance whose class is being predicted. For example, a selection function could be one that chooses the base classifier that performs best on the training data instances that are most like the testing data instance being considered for classification. We currently consider fusion methods only throughout the remainder of this paper.

The combinations functions we consider take as input one of three different types of output from the base classifiers.

- *Labels.* Functions that make use of the most likely class to which an instance belongs.
- *Label rankings.* Functions that make use of ranked lists of class label predictions (e.g., sorted by likelihood or probability that the instance belongs to the class).
- *Measurements.* Functions that make use of vectors of length k (i.e., the number of classes), where element j corresponds to a measurement of an instance associated with class j. These measurements are often intrinsic values computed within each classification method. In this work, the measurements are the probabilities that an instance belongs to a particular class.

A variety of fusion methods have been developed. We discuss several here that are used in the experiments described in Section 4. An example of a fusion function that uses label outputs is the majority voting function [19]. In majority voting, the labels output from each base classifier are used as votes on the predicted class, and the class with the most votes becomes the predicted class of the ensemble classifier. The sum rule is an example of a fusion function that uses measurement outputs from the base classifiers [18]. The sum rule first sums the measurement output vectors from the base classifiers for a given test instance, and then chooses the class corresponding to the largest sum as the predicted class. The linear combination rule is identical to the sum rule except that each of the base classifiers is assigned a weight, which is used to scale the measurement vectors before summing [26]. In our experiments, we determined the weights using the least squares method to maximize training accuracy. In stacking, the output vectors are treated as input to a new classification problem, where a classification algorithm, such as the SVM classification algorithm, builds a model to act as the fusion function. We have also been alerted, by a referee, to an ensemble method that is implemented inside of Weka, called Ensemble Selection[8]. In the future we would like to explore this method and compare our results with it.

The process of creating ensemble classifier models involves two major steps. First, the base classifiers are created during a generation phase, and then the models are combined during a combination phase. The goal of the generation phase is to create a diverse, accurate set of base classifiers. A recent survey on several diversity measures

[5] illustrates the challenges associated with creating a diverse set of base classifiers, and accuracy will be discussed in Section 3.1. Examples of methods for generating diverse base classifiers include sampling the training data (e.g., bagging [6]) and sampling the feature space (e.g., the random subspace method [15]). Another example method, the method of random forests [7], combines these two sampling strategies.

For some homogeneous ensemble models (e.g., models using a linear combination fusion function), several model parameters need to be learned or fit. Typically, the training of the ensemble model parameters is performed using the training data set. However, there are issues with such an approach [13], and more work in this area is required to better understand the implications of such a training strategy. Such work is beyond the scope of this paper, but will be pursued in future work.

2.2. Heterogeneous Ensemble Classifiers. A heterogeneous ensemble is an ensemble with a set of base classifiers that consist of models created using different algorithms. The same combination functions that are used to create homogeneous ensembles can be used to create heterogeneous ensembles. The main difference lies in the methods used for creating the set of base classifier. The methods available for creating base classifier sets for homogeneous ensembles are modified so that models built from different classification algorithms can be combined to form a set of base classifiers. Currently, there is no clear choice on how to combine these base classifiers most effectively. Furthermore, there are open questions regarding which base classifiers to use and how they should be combined for optimal performance.

Motivation. Using different types of base classifiers leads to diversity in the same way that changing model parameters can in creating homogeneous ensemble classifiers. Different base classifier types can have different internal representations and may be biased in different ways. This leads to classifiers that will disagree with each other to some extent over a set of data instances covering a wide range of the feature space. This disagreement between the base classifiers is essential for the success of an ensemble classifier and is what we refer to as diversity. Without diversity in the base classifier models, there is no point in using an ensemble, as the output of the ensemble classifier will be identical to the output of each of the base classifiers. On the other hand, we would also like the base classifiers to be as accurate as possible. We do not want to force diversity in such a way that we end up with base classifiers that have too much error or that do not generalize well. Using a heterogeneous set of base classifiers, then, is a way to introduce diversity while keeping accuracy high.

Diversity. Table 2.1 illustrates how different base classifier algorithms can lead to diversity in an ensemble. These classifiers were trained using the same data set. They were then tested using a testing data set and their outputs were recorded. The differences in their outputs represent the extent to which they "disagree" about the probability distributions for the test instances.

2.2.1. HEMLOCK. Hemlock has been designed to create and test heterogeneous ensemble models. As such it contains methods for creating base classifiers, applying fusion functions, and evaluating classifier models. Hemlock currently uses interfaces to Weka classification algorithms for creating base classifiers. Input and output is passed to and from Hemlock via XML. The input, or experiment, files contain information about which models to use, model parameters, evaluation methods, and data sets to be used in an experiment. Ranges of model parameters can be specified as well, leading to collections of experiments, each corresponding to a particular set of parameters allowed. Currently only full factorial experiments for a given set of parameter values (i.e., sets of experiments where all possible combinations

	Naive Bayes		*Decision Tree*	
Instance	*Measurement*	*Label*	*Measurement*	*Label*
1	$[0.99, 0.01]$	1	$[0.87, 0.13]$	1
2	$[0.00, 1.00]$	2	$[0.18, 0.82]$	2
3	$[0.04, 0.96]$	2	$[0.18, 0.82]$	2
4	$[0.01, 0.99]$	2	$[0.87, 0.13]$	1
5	$[0.00, 1.00]$	2	$[0.18, 0.82]$	2
6	$[0.00, 1.00]$	2	$[0.05, 0.95]$	2
7	$[0.99, 0.01]$	1	$[0.87, 0.13]$	1
8	$[1.00, 0.00]$	1	$[0.05, 0.95]$	2

TABLE 2.1

Measurement outputs from two classifiers trained on the same data. The differences in these outputs for the same instances is what we refer to as diversity and is essential for creating ensemble classifiers. In this example the diversity is introduced by using different classification algorithms.

of the given parameters values are tested) are supported. Several standard evaluation methods have been implemented in HEMLOCK as well: e.g., stratified k-fold cross validation, ROC curve generation, and generalization error calculations. Creation of base classifiers using a particular set of parameters or via random sampling can be specified as well. HEMLOCK currently includes the fusion methods of majority voting, the sum rule, and a linear combination rule (where the weights are computed using linear l_2 regression).

3. Evaluation of classification algorithms. The evaluation of performance or accuracy of a classification algorithm is not necessarily straightforward. Real world problems often have different sets of (potentially conflicting or competing) requirements. Hence, an algorithm (or more precisely, a particular parametrized instance of an algorithm) that may work well in solving one classification problem may perform poorly on other problems. Some issues that motivate different approaches to classifier evaluation include model interpretability, predictive ability of models created (including accuracy and data overfitting), and computational time/effort required during both the training and application of a model.

3.1. Evaluation Considerations.

Interpretability. Sometimes the models from classification algorithms need to be interpretable by a human being, e.g., for explanatory analysis of classes in addition to prediction. It may be that the user wants to ensure the sanity of the model, or it may be that he or she wants to learn something about the underlying classes by studying the models. Some classification algorithms create models that are easy and natural for humans to interpret. Ensembles models, however, are generally not easy to interpret, even when the base classifiers individually are easy to interpret.

Accuracy. Generalization error is a measure of how well a model predicts the classes for a set of instances it has never seen before (i.e., a testing data set). Training error is a measure of how well a model predicts classes for the same set of instances that were used to train the model. Training error is a very optimistic estimation of the true model error over the entire instance space, whereas generalization error is more pessimistic and a less biased estimate of the true error. Accuracy then usually refers to the complement of the generalization error.

Overfitting. One of the reasons that training error is regarded as optimistic is based on data overfitting. Training error does not incorporate the extent a classifi-

cation model has overfit the data. If a model is too specific to the training data and does not generalize well to the entire instance space, then it has overfit the training data. There are various strategies that classification algorithms can employ to ensure that resulting models both perform well on the training data and generalize to unseen training instances. For example, many decision tree classification algorithms contain methods for pruning of tress to avoid singleton leaf nodes, which typically do not generalize well.

Computational Time. There are two considerations associated with the computational running time of a classification algorithm. One is the amount of time it takes to build the models, called training time. The other is the time it takes to predict the class of an instance, called prediction time. Some algorithms require very little time to train but more time to predict, and some vice versa. For example, artificial neural networks [22] (not currently part of HEMLOCK) can take a very long time to train, but the models they produce are a simple linear combination of the features, leading to constant prediction time complexity. On the other hand, a nearest neighbor classifier spends no time on training, but during classification the training data set must be searched to find the nearest neighbors for each testing instance, which can be computationally expensive. A less extreme example are decision trees, which for many methods can be trained in $O(mn \log n)$ time and tested in $O(\log_s n)$ time, where s is the minimum number of splits allowed over all the attributes (e.g., $s = 2$ in the case of binary decision trees induced from data with only continuous attributes) [12]. Another consideration that impacts computational time is how much and to what extent the training and testing methods of a classification algorithm parallelizes. In general, ensemble methods can be parallelized well because the ensemble members can be trained in parallel, and the ensemble predictions can be executed in parallel as well.

3.2. Evaluation Measures. The following descriptions of evaluation measures assume we are solving a two-class classification problem and that the true class labels are known for the testing data instances. All of the following concepts can be generalized to multi-class problems but we do not, in this exposition, for the sake of clarity. When dealing with two-class problems the class labels are "positive" (typically referring to the class in which we are most interested in classifying) and "negative". Table 3.1 lists definitions used throughout this section.

Value	*Symbol*	*True Label*	*Predicted Label*
True positives	TP	positive	positive
True negatives	TN	negative	negative
False positives	FP	negative	positive
False negatives	FN	positive	negative

TABLE 3.1
Definitions used in measures for evaluating two-class classifier models.

Confusion Matrices. A confusion matrix can be created from a model f and a set of instances with known true class labels D_{te}, and reflects how well a classifier correctly classifies those instances per class. An example confusion matrix for the two-class classification problem is as follows.

		Predicted Class	
		Positive	Negative
True Class	Positive	*TP*	*FN*
	Negative	*FP*	*TN*

Rows of the matrix are associated with true labels, and columns represent the predicted labels. Hence, the sum of the row values equals the number of instances in D_{te} that have the true class corresponding to that row. Similarly the sum of the column values equals the number of instances in D_{te} where $f(x)$ has predicted the class corresponding to that column. It is easy to see that elements on the diagonal contain the counts of instances that the model correctly labeled for each class. With a two class classification problem, you always get a 2×2 confusion matrix where the values in the matrix correspond to TP, FN, FP, TN from left to right and top to bottom. TP and TN are on the diagonal entries and correspond to correctly classified instances.

Accuracy. We denote the accuracy of a classifier model, f, as

$$A(f) = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \ . \tag{3.1}$$

As mentioned above, it is important to realize the importance of choosing a set of instances for calculating accuracy. Accuracy can be calculated using any set of instances, but a data set containing instances different than those used in training the model begin evaluated will be less biased.

Receiver Operator Characteristic (ROC) Curve. ROC curves depict the potential of a model for correct classification in two-class problems. To generate an ROC curve for a classifier model, the set of continuous outputs relative to the positive class—i.e., the values $f(\mathbf{x}), \mathbf{x} \in \mathcal{D}_{te}$ are first sorted in descending order. The ROC curve is then a plot of the true positive rate, $TP/(TP+FN)$, as a function of the false positive rate, $FP/(FP+TN)$, computed using each of the sorted outputs as a cutoff threshold for labeling instances as positive. The true positive rate is also called the *sensitivity* of a classifier model, and the false positive rate is the complement of the *specificity* of a classifier model. Thus, an ROC curve is sometimes referred to as a plot of sensitivity versus (1 - specificity).

ROC analysis can also be used to compare models built for multi-class classification problems. In the simplest case, each model will correspond to one point on the graph. One can then easily compare different properties of the models based on their placement on the graph. [14]

Area Under the Curve (AUC). The AUC measurement corresponds to the ROC analysis for two-class classification models. The area under the ROC curve described above can be calculated to give a scalar representation of the most important aspect of the plot: the potential of the classifier to perform well in separating regions of feature space by class.

3.3. Validation Methods. Methods for validating the performance of classifier models typically employ a scheme for choosing training and testing data sets and a process for determining the significance of the results [19, 23].

Holdout. The simplest method to validate a classifier model is called the holdout method and simply involves separating a given set of instances with known labels into two sets, $\mathcal{D}_{tr}$ and $\mathcal{D}_{te}$. The first set is used to train the model and the second set is used to test the model. This method is not often used because it is difficult to determine the significance of the tests results since it is based on a single split of

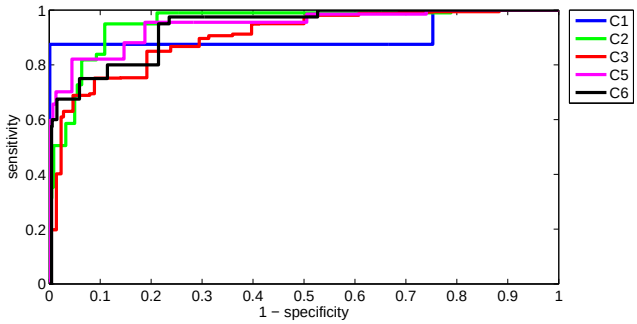

FIG. 3.1. *Example ROC curves for random tree classifiers trained on the "anneal" data set (6 classes, 5 classes represented in test data). The different curves correspond to ROC curves for the two-class problems of one class versus the rest.*

the data only. However, the method is quick and does indicate a rough estimate of performance.

Stratified k-Fold Cross Validation. The method of k-fold cross validation is used to compute generalization error and determine the significance of the testing results. The methods starts by first dividing the training data into k partitions, or folds, where each fold contains instances with approximately equal class distribution (i.e., are stratified). If the folds are not stratified, the test results may poorly approximate the generalization error. Each of the folds will be used as a testing set for exactly one model trained using the remaining folds as training data. Once the models are created, they are tested and the results are averaged to determine generalization error and then analyzed statistically to determine the significance of the results. A common value used for k is 10, but their may be advantages for using other values depending on the problem at hand. One disadvantage of using large values of k is that a large number of models will need to be created and tested; this may not be practical due to the computational time required.

Other related methods are the leave-one-out method and 5×2-fold cross validation. In the leave-one-out method, each of the instances is used individually as a testing set with the remaining instances used as the training set. With n instances, this is equivalent to n-fold cross validation. 5×2-fold cross validation consists of 2-fold cross validation performed five times, with the results being averaged across the five runs [10].

Bootstrapping (Random Fold) Validation. Bootstrapping is similar to k-fold cross validation in that different models are created and tested, with the performance results computed as averages over all the models. The training sets are created by randomly selecting instances with replacement. All of the instances that are not chosen for the training set are used as the test set. Because the instances are drawn with replacement, a larger training set can be created than with k-fold cross validation. However, many of the instances in the training set may be duplicates. The test results of this method should be meaningful, as the class distributions of the test set will likely match the class distribution of the original set of instances and the training set because of the random way the set is chosen. This method can be useful when only a small number of instances with known labels are available.

4. Numerical Experiments. In this section we describe the experiments we performed on base classifiers, homogeneous ensemble classifiers, and heterogeneous ensemble classifiers.

4.1. Data. The data we used in our experiments is the data from [3]. We specifically only used data that had no unknown attributes. See Table B.1 for a summary of the data sets used in the experiments.

4.2. Experiments. First we performed parameter sweeps for the base classifier algorithms to find sets of parameters that performed well. We measured the performance by using 2-fold cross validation to calculate accuracy and then taking the average of those accuracies across all of the data sets. We then used the parameter combinations to construct homogeneous ensemble classifiers. Each of these homogeneous classifiers used the top two-thirds of the parameter combinations of the base classifier algorithm. This meant that the homogeneous ensembles had different numbers of base classifiers as there were different numbers of total parameters combinations for each of the base classifier types. We also created heterogeneous ensembles by using all of the base classifier combinations that were used in the homogeneous ensembles. Each of the ensembles were combined using either voting or the sum rule.

We then repeated the same strategy for creating creating ensembles, with the additional use of bagging. Each of the bagged ensembles were constructed using 200 base classifiers trained on a bag with the same size as the original training set. We evaluated the ensembles by measuring the accuracy using 2-fold cross validation for each of the data sets and then averaging over the all of the data sets.

Classification		***Fusion***
Type	*Algorithm*	*Function*
Decision Tree	Random Tree (RT) [7]	Voting [19]
Probabilistic	Naive Bayes (NB) [16, 20]	Sum Rule [19]
Function	Support Vector Machine (SVM) [17, 21, 2]	
Instance Based	k Nearest Neighbor (KNN) [1]	
Rule Based	Ripper [9, 23]	

TABLE 4.1
The classification algorithms and fusion functions used in HEMLOCK *to create ensembles.*

4.3. Results. As was expected, on average, the ensembles performed better than the base classifiers in terms of accuracy. For the averaged results corresponding to the ensemble classifiers, the only clear trend was the improvement due to the use of the sum rule over voting when not using bagging, as can be seen in Figure 4.1. Surprisingly, the results were not as clear for bagging. Figure 4.2 shows no clear correlation between bagging and accuracy. The difference in accuracies for the different ensemble methods is not clear when averaged over all the data sets either. The heterogeneous ensembles and the decision tree ensembles result in higher averaged accuracies, but the differences may not be significant. More work on analyzing this data is needed.

The accuracies for each data set and the averaged accuracies are presented in Appendix C. When the accuracies are averaged over all data sets, it is difficult to see any advantages over using one ensemble method over the other. However when looking at the details of how these methods perform on particular data sets, there are often large differences in accuracy. This suggests that exploring these different ensemble methods may lead to better understanding of the behavior of the different methods.

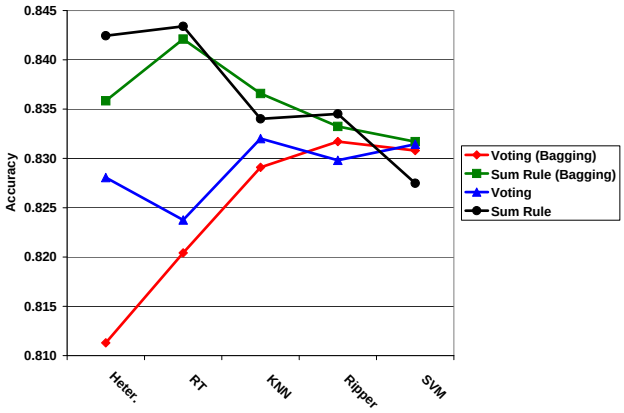

FIG. 4.1. *The average accuracies over all data sets showing that ensembles that use the sum rule do better than those that use voting.*

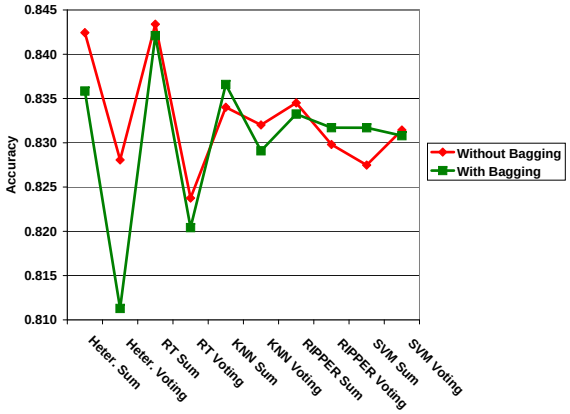

FIG. 4.2. *The average accuracies over all data sets, comparing the use of bagging for each ensemble method.*

However, the ensemble generation techniques need to be fine tuned to take advantage of information known about the data set for which we are building a classifier model.

We did notice that when the performance of a homogeneous ensemble was poor, relative to the other homogeneous ensembles trained on the same data set, then the heterogeneous ensemble also had lower performance. Examples of this can be seen in Figure 4.3. There are points in this figure where the random tree ensembles perform much worse than the average homogeneous ensemble, leading to mediocre heterogeneous ensemble performance. This suggests that if one of the homogeneous ensembles performs poorly, we should not include the associated base classifier type into the heterogeneous ensemble, or we should include less base classifiers of that type. More work is needed to see how well this idea generalizes to other data sets.

Some of the results in Figure 4.3 indicate that all the ensemble methods performing equally well for a particular data set. However, the results for the *letter* data set illustrate that heterogeneous ensembles can outperform the homogeneous ensembles. In this work there was no systematic method used for selecting what balance of base

classifiers to use, and this result indicates that paying closer attention to such a detail may be important.

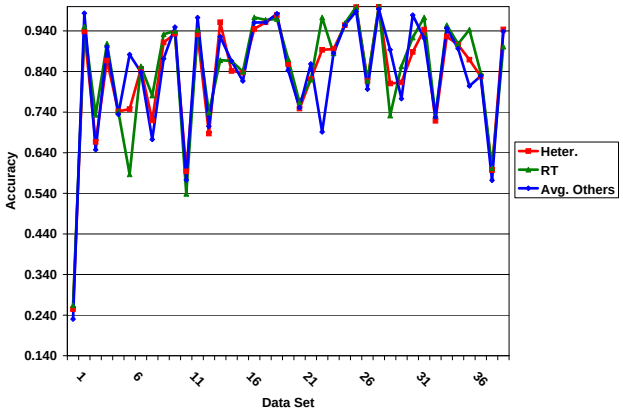

FIG. 4.3. *Accuracy of the heterogeneous ensemble, random tree ensemble, and the average for the rest of the homogeneous ensembles, for each data set.*

5. Future Work. This work was an exploratory experiment in learning more about creating heterogeneous ensembles and how they can be used effectively. The majority of the work went into creating the HEMLOCK framework, which allows us to experiment with these methods in future work.

We need to create better methods for training our ensembles for the data sets in which we are interested. The training sets may include indicators which will allow us to fine tune the way the base classifiers are generated or to help determine which ensemble techniques to use. This will be the main focus of future work. We would also like to determine, in general, whether the strategies used for training homogeneous ensembles should be used for training heterogeneous ensembles as well. It will be important to explore which types of base classifiers lead to better ensemble performance. This could result in either general guidelines or a set of new methods.

We would also like to explore novel methods that take advantage of the unique nature of heterogeneous classifiers. The different base classifier algorithms all have their own strengths, and it would be interesting to try to use each of the base classifiers in ways that take advantage of those strengths. For example, if a base classifier works better with nominal features, then it could be trained on a subspace of the original training set with just the nominal features.

6. Conclusions. Our initial attempt at creating heterogeneous ensembles did not lead to significant gains in classifier performance. However, the results presented here will serve as a good benchmark for evaluating performance of ensemble classifiers. We have demonstrated that heterogeneous ensembles perform better than homogeneous ensembles in some cases, but more work is needed to better understand under what circumstances the use of heterogeneous ensemble classifiers leads to this improvement. Using the HEMLOCK software framework developed in this work, we can now investigate and evaluate new methods for heterogeneous ensemble classification, and we intend to follow up on the many questions identified in work presented here.

We also found that the sum rule performs better than voting on average. We expected this because the sum rule has more information from the base classifiers,

since it uses measurement values. This result has lead us to believe that it is worth while to continue exploring fusion functions that use measurement values.

7. Acknowledgments. We would like to thank Philip Kegelmeyer for providing the data sets for our testing and for helpful suggestions throughout the project. We also thank the developers of the WEKA and JAMA libraries used in HEMLOCK.

This project was made possible from funding by the LDRD Program at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energys National Nuclear Security Administration under Contract DE-AC04-94AL85000.

REFERENCES

[1] D. AHA AND D. KIBLER, *Instance-based learning algorithms*, Machine Learning, 6 (1991), pp. 37–66.
[2] M. AIZERMAN, E. BRAVERMAN, AND L. ROZONOER, *Theoretical foundations of the potential function method in pattern recognition learning*, Automation and Remote Control, 25 (1964), pp. 821–837.
[3] R. BANFIELD, L. HALL, K. BOWYER, AND W. P. KEGELMEYER, *A comparison of decision tree ensemble creation techniques*, IEEE Trans. Pat. Recog. Mach. Int., 29 (2007), pp. 173–180.
[4] J. BASILICO, D. DUNLAVY, S. VERZI, T. BAUER, AND W. SHANEYFELT, *Yucca mountain LSN archive assistant*, Tech. Rep. SAND2008-1622, Sandia National Laboratories, 2008.
[5] S. BIAN AND W. WANG, *On diversity and accuracy of homogeneous and heterogeneous ensembles*, Intl. J. Hybrid Intel. Sys., 4 (2007), pp. 103–128.
[6] L. BREIMAN, *Bagging predictors*, Machine Learning, 24 (1996), pp. 123–140.
[7] L. BREIMAN, *Random forests*, Machine Learning, 45 (2001), pp. 5–32.
[8] R. CARUANA, A. NICULESCU-MIZIL, G. CREW, AND A. KSIKES, *Ensemble selection from libraries of models*, in Proc. ICML, 2004.
[9] W. W. COHEN, *Fast effective rule induction*, in Twelfth International Conference on Machine Learning, Morgan Kaufmann, 1995, pp. 115–123.
[10] T. G. DIETTERICH, *Approximate statistical tests for comparing supervised classification learning algorithms*, Neural Comput., 10 (1998), pp. 1895–1923.
[11] T. G. DIETTERICH, *Ensemble methods in machine learning*, in Proc. International Workshop on Multiple Classifier Systems, 2000, pp. 1–15.
[12] R. O. DUDA, P. E. HART, AND D. G. STORK, *Pattern Classification*, Wiley-Interscience, 2nd ed., 2000.
[13] R. P. W. DUIN, *The combining classifier: To train or not to train*, in Proc. International Conference on Pattern Recognition, vol. 2, 2002, pp. 765–770.
[14] T. FAWCETT, *An introduction to roc analysis*, Pattern Recogn. Lett., 27 (2006), pp. 861–874.
[15] T. K. HO, *The random subspace method for constructing decision forests*, IEEE Trans. Pattern Anal. Mach. Intell., 20 (1998), pp. 832–844.
[16] G. H. JOHN AND P. LANGLEY, *Estimating continuous distributions in bayesian classifiers*, in Eleventh Conference on Uncertainty in Artificial Intelligence, 1995, pp. 338–345.
[17] S. S. KEERTHI, S. K. SHEVADE, C. BHATTACHARYYA, AND K. R. K. MURTHY, *Improvements to platt's smo algorithm for svm classifier design*, Neural Comput., 13 (2001), pp. 637–649.
[18] J. KITTLER, M. HATEF, R. P. W. DUIN, AND J. MATAS, *On combining classifiers*, IEEE Trans. Pattern Anal. Mach. Intell., 20 (1998), pp. 226–239.
[19] L. I. KUNCHEVA, *Combining Pattern Classifiers: Methods and Algorithms*, Wiley-Interscience, 2004.
[20] T. M. MITCHELL, *Machine Learning*, McGraw-Hill, New York, 1997.
[21] J. C. PLATT, *Fast training of support vector machines using sequential minimal optimization*, in Advances in Kernel Methods: Support Vector Learning, MIT Press, Cambridge, MA, USA, 1999, pp. 185–208.
[22] S. RUSSELL AND P. NORVIG, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, 2nd ed., 2003.
[23] P.-N. TAN, M. STEINBACH, AND V. KUMAR, *Introduction to Data Mining*, Addison Wesley, May 2005.
[24] W. WANG, D. PARTRIDGE, AND J. ETHERINGTON, *Hybrid ensembles and coincident failure diversity*, in Proc. International Joint Conference on Neural Networks, 2001.

[25] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*, Morgan Kaufmann, June 2005.
[26] D. H. Wolpert, *Stacked generalization*, Neural Netw., 5 (1992), pp. 241–259.

Appendix A. Base Classifiers. A base classifier refers to a single classifier model whose output is used as input for an ensemble classifier. The base classifiers used in this work consist of several typical learning models used in ensemble classifier models. The base classifiers chosen are representative of the major strategies currently used in solving many classification problems: decision trees, probabilistic models, functional models, instance-based models, and rule-based models.

Decision Trees. Decision tree algorithms create models using a divide and conquer strategy to recursively partition the feature space along feature axes until "pure" partitions (i.e., partitions containing data from a single class only) are found. Choices for a partitioning strategy (i.e., how to choose features and split values), stopping criteria for growing trees (i.e., if and when to stop growing a tree before pure partitions are found), and a partition aggregation strategy (i.e., how to prune a tree to avoid overfitting the data) lead to different decision tree methods. Also, sampling from feature space at each decision node leads to *random decision trees*. For small data sets, the resulting decision trees are often easy to interpret and analyze, thus making decision trees a popular choice for data analysts [7].

Bayesian Classifiers. Bayesian classifiers consist of a probability model for each class combined with a decision rule for choosing the class for a given data instance. The probability model is a conditional model that is estimated using Bayes' Theorem:

$$p(y_i|\mathbf{x}_i) = \frac{p(y_i)\ p(\mathbf{x}_i|y_i)}{p(\mathbf{x}_i)} \equiv \text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}} \ . \tag{A.1}$$

The naive Bayes classifier is a particular Bayesian classifier which includes the assumption that the features are conditionally independent:

$$p(\mathbf{x}_i|y_i) = p(\langle a_1(\mathbf{x}_i), \ldots, a_m(\mathbf{x}_i)\rangle|y_i) = \prod_{j=1}^{m} p(a_j(\mathbf{x}_i)|y_i) \ . \tag{A.2}$$

A common decision rule for Bayesian classifiers is to choose the class that is most probable (i.e., the *maximum a posteriori*, or MAP, decision rule) [16, 20].

Support Vector Machines. Support vector machines (SVMs) [17, 21] are linear classifiers designed to find a hyperplane which simultaneously minimizes classification error and maximizes the distance, or margin, between the hyperplane and data instances from two different classes. Extensions of SVMs used in the work presented here include methods for handling misclassifications (using soft, or relaxed, margins) and for embedding data into higher dimensional feature spaces in order to estimate nonlinear decision boundaries in the original feature space (i.e., the *kernel trick* [2]).

Nearest Neighbor Classifiers. Nearest neighbor classifiers [1] are examples of classifiers that do not need training. To label a data instance, a nearest neighbor classifier examines the instances in the training set that are closest to it in terms of feature similarity or distance in some metric space and predicts a label based on the labels of those "neighboring" instances. The class boundaries are therefor implicit, often leading to more flexible decision boundaries than some of the other explicitly formed boundaries discussed in this section. Choices for the number of neighbors, the similarity/distance measure, and voting/weighting schemes for combining information (attributes, labels, etc.) from neighbors lead to different variants of nearest neighbor classifiers.

Association Rules. Models created from these classifier methods are sets of rules consisting of logical conjunctions of decision boundaries along feature axes. An example of a rule is

$$\{a_1 = \text{“scalar”}\} \wedge \{a_2 > 1.0\} \implies y_1 \, . \tag{A.3}$$

These rules are often built in a general to specific manner, where new conditions are added to the conjunction as long as the rule continues to improve classifier performance (see Section 3.2 for more information on classifier performance). An opposing approach, a specific to general rule building strategy, starts with a specific rule targeting some instance in the training set and then proceeds by removing conditions from the conjunction while the rule continues to improve classifier performance [9, 23].

Appendix B. Data Sets Used in Experiments.

#	*Name*	*Instances*	*Classes*	*Continuous Attributes*	*Nominal Attributes*
1	abalone	4177	29	7	1
2	anneal	898	6	6	32
3	bupa	345	2	6	0
4	car	1728	4	0	6
5	credit-g	1000	2	7	13
6	dna	3186	3	0	180
7	ecoli	326	8	7	0
8	glass	214	6	9	0
9	ion	351	2	34	0
10	iris	150	3	4	0
11	krk	28056	18	6	0
12	krkp	3196	2	0	36
13	led-24	5000	10	0	24
14	letter	36000	26	16	0
15	lrs	530	10	93	0
16	lymph	148	4	3	15
17	nursery	12961	5	0	8
18	page	5473	5	10	0
19	pendigits	10993	10	16	0
20	phoneme	5404	2	5	0
21	pima	768	2	8	0
22	promoters	106	2	0	57
23	ringnorm	300	2	20	0
24	sat	6435	6	36	0
25	segment	2310	7	19	0
26	shuttle	58000	7	9	0
27	sonar	208	2	60	0
28	soybean-small	47	4	0	35
29	splice	3190	3	0	60
30	threenorm	300	2	20	0
31	tic-tac-toe	958	2	0	9
32	twonorm	300	2	20	0
33	vehicle	846	4	18	0
34	vote	435	2	0	16
35	vote1	435	2	0	15
36	vowel	528	11	10	0
37	waveform	5000	3	21	0
38	yeast	1484	10	8	0
39	zip	9298	10	256	0

TABLE B.1
Meta information for each data set used in the experiments.

Appendix C. Numerical Results.

Data Set	*Heter.*	*RT*	*KNN*	*RIPPER*	*SVM*
abalone	0.258	0.267	0.200	0.211	0.264
anneal	0.958	0.967	0.983	0.982	0.986
bupa	0.698	0.733	0.638	0.681	0.623
car	0.900	0.911	0.916	0.862	0.924
credit-g	0.734	0.735	0.735	0.733	0.743
dna	0.626	0.589	0.801	0.936	0.921
ecoli	0.834	0.859	0.840	0.853	0.847
glass	0.771	0.793	0.676	0.700	0.635
ion	0.929	0.920	0.846	0.900	0.889
iris	0.940	0.933	0.947	0.933	0.955
krk	0.550	0.538	0.687	0.747	0.287
krkp	0.946	0.940	0.962	0.992	0.970
led-24	0.736	0.741	0.624	0.744	0.750
letter	0.914	0.871	0.982	0.954	0.845
lrs	0.864	0.861	0.877	0.847	0.885
lymph	0.858	0.864	0.819	0.758	0.824
nursery	0.965	0.971	0.978	0.972	0.930
page	0.966	0.968	0.960	0.971	0.951
pendigits	0.972	0.971	0.993	0.974	0.980
phoneme	0.868	0.868	0.893	0.861	0.772
pima	0.747	0.766	0.724	0.754	0.768
promoters	0.906	0.858	0.811	0.821	0.908
ringnorm	0.953	0.963	0.563	0.733	0.717
sat	0.888	0.885	0.910	0.883	0.864
segment	0.963	0.965	0.964	0.965	0.930
shuttle	0.999	0.999	0.999	1.000	0.966
sonar	0.769	0.841	0.851	0.761	0.727
soybean-small	1.000	1.000	1.000	0.981	1.000
splice	0.742	0.726	0.813	0.950	0.924
threenorm	0.853	0.860	0.783	0.657	0.830
tic-tac-toe	0.902	0.896	0.977	0.979	0.983
twonorm	0.960	0.973	0.950	0.827	0.963
vehicle	0.729	0.719	0.691	0.701	0.752
vote	0.938	0.945	0.936	0.956	0.952
vote1	0.906	0.899	0.903	0.901	0.920
vowel	0.922	0.956	0.975	0.752	0.684
waveform	0.844	0.837	0.800	0.814	0.865
yeast	0.621	0.606	0.551	0.595	0.590
zip	0.924	0.899	0.967	0.907	0.949
mean	0.842	0.843	0.834	0.835	0.827
std	0.148	0.149	0.166	0.151	0.171

TABLE C.1

Accuracy results for ensembles without bagging using the sum rule as the fusion function.

Data Set	*Heter.*	*RT*	*KNN*	*RIPPER*	*SVM*
abalone	0.270	0.262	0.221	0.188	0.261
anneal	0.941	0.915	0.979	0.982	0.986
bupa	0.733	0.730	0.612	0.684	0.641
car	0.840	0.807	0.920	0.863	0.928
credit-g	0.721	0.717	0.737	0.727	0.752
dna	0.569	0.542	0.790	0.931	0.924
ecoli	0.859	0.849	0.840	0.807	0.864
glass	0.762	0.754	0.665	0.686	0.644
ion	0.917	0.920	0.843	0.900	0.866
iris	0.953	0.939	0.961	0.913	0.966
krk	0.464	0.467	0.695	0.715	0.284
krkp	0.884	0.861	0.953	0.994	0.949
led-24	0.660	0.631	0.648	0.744	0.747
letter	0.828	0.782	0.970	0.943	0.845
lrs	0.870	0.863	0.870	0.845	0.888
lymph	0.838	0.840	0.839	0.737	0.845
nursery	0.959	0.967	0.979	0.969	0.931
page	0.966	0.967	0.958	0.972	0.950
pendigits	0.970	0.964	0.993	0.972	0.980
phoneme	0.852	0.850	0.882	0.860	0.758
pima	0.762	0.759	0.734	0.758	0.779
promoters	0.820	0.802	0.792	0.803	0.906
ringnorm	0.937	0.967	0.547	0.754	0.730
sat	0.882	0.881	0.908	0.880	0.867
segment	0.951	0.952	0.956	0.968	0.934
shuttle	0.999	0.999	0.999	1.000	0.966
sonar	0.788	0.822	0.822	0.755	0.765
soybean-small	1.000	1.000	1.000	0.981	1.000
splice	0.848	0.814	0.780	0.947	0.919
threenorm	0.830	0.843	0.850	0.694	0.854
tic-tac-toe	0.872	0.874	0.979	0.973	0.983
twonorm	0.960	0.977	0.940	0.817	0.960
vehicle	0.736	0.723	0.705	0.686	0.748
vote	0.945	0.952	0.929	0.954	0.963
vote1	0.922	0.910	0.897	0.894	0.915
vowel	0.848	0.920	0.913	0.769	0.710
waveform	0.838	0.828	0.807	0.821	0.871
yeast	0.601	0.608	0.573	0.582	0.596
zip	0.900	0.866	0.963	0.897	0.950
mean	0.828	0.824	0.832	0.830	0.831
std	0.151	0.155	0.161	0.154	0.169

TABLE C.2

Accuracy results for ensembles without bagging using voting as the fusion function.

Data Set	*Heter.*	*RT*	*KNN*	*RIPPER*	*SVM*
abalone	0.255	0.264	0.210	0.211	0.270
anneal	0.939	0.951	0.983	0.987	0.981
bupa	0.667	0.733	0.629	0.684	0.629
car	0.867	0.909	0.915	0.852	0.933
credit-g	0.742	0.742	0.745	0.717	0.742
dna	0.748	0.587	0.803	0.923	0.919
ecoli	0.846	0.853	0.844	0.807	0.864
glass	0.720	0.781	0.691	0.678	0.649
ion	0.912	0.932	0.852	0.889	0.874
iris	0.933	0.940	0.961	0.935	0.953
krk	0.594	0.539	0.686	0.748	0.285
krkp	0.931	0.943	0.958	0.992	0.969
led-24	0.687	0.739	0.626	0.742	0.747
letter	0.962	0.868	0.983	0.951	0.843
lrs	0.841	0.866	0.874	0.834	0.889
lymph	0.838	0.840	0.846	0.746	0.858
nursery	0.945	0.973	0.978	0.972	0.932
page	0.961	0.967	0.960	0.972	0.950
pendigits	0.979	0.970	0.993	0.974	0.980
phoneme	0.858	0.870	0.894	0.862	0.774
pima	0.749	0.764	0.720	0.758	0.775
promoters	0.821	0.820	0.821	0.830	0.925
ringnorm	0.894	0.973	0.567	0.780	0.727
sat	0.895	0.885	0.907	0.884	0.864
segment	0.954	0.959	0.966	0.965	0.932
shuttle	0.999	0.999	0.999	1.000	0.966
sonar	0.817	0.817	0.851	0.774	0.764
soybean-small	1.000	1.000	1.000	0.981	1.000
splice	0.810	0.732	0.810	0.947	0.924
threenorm	0.813	0.853	0.800	0.663	0.856
tic-tac-toe	0.888	0.924	0.973	0.981	0.983
twonorm	0.943	0.973	0.960	0.840	0.967
vehicle	0.719	0.730	0.714	0.698	0.773
vote	0.926	0.954	0.926	0.959	0.954
vote1	0.906	0.908	0.906	0.890	0.894
vowel	0.869	0.943	0.972	0.759	0.682
waveform	0.828	0.837	0.802	0.818	0.866
yeast	0.597	0.602	0.534	0.591	0.592
zip	0.944	0.901	0.966	0.902	0.948
mean	0.836	0.842	0.837	0.833	0.832
std	0.143	0.150	0.165	0.151	0.169

TABLE C.3

Accuracy results for ensembles with bagging using the sum rule as the fusion function.

Data Set	*Heter.*	*RT*	*KNN*	*RIPPER*	*SVM*
abalone	0.261	0.264	0.226	0.186	0.265
anneal	0.914	0.918	0.971	0.978	0.980
bupa	0.649	0.730	0.591	0.643	0.649
car	0.841	0.819	0.917	0.867	0.933
credit-g	0.725	0.718	0.742	0.718	0.758
dna	0.614	0.543	0.786	0.930	0.923
ecoli	0.847	0.841	0.863	0.811	0.862
glass	0.715	0.780	0.638	0.654	0.659
ion	0.900	0.931	0.843	0.903	0.878
iris	0.947	0.940	0.960	0.960	0.967
krk	0.452	0.465	0.693	0.706	0.283
krkp	0.880	0.849	0.952	0.990	0.951
led-24	0.625	0.643	0.653	0.740	0.743
letter	0.876	0.785	0.970	0.946	0.845
lrs	0.862	0.863	0.871	0.848	0.890
lymph	0.791	0.843	0.830	0.770	0.819
nursery	0.946	0.969	0.978	0.968	0.931
page	0.958	0.967	0.957	0.973	0.951
pendigits	0.964	0.965	0.992	0.973	0.980
phoneme	0.849	0.857	0.884	0.865	0.757
pima	0.714	0.740	0.743	0.738	0.766
promoters	0.735	0.698	0.736	0.886	0.935
ringnorm	0.867	0.953	0.550	0.760	0.720
sat	0.881	0.880	0.910	0.878	0.865
segment	0.935	0.959	0.957	0.964	0.935
shuttle	0.998	0.999	0.999	1.000	0.966
sonar	0.789	0.808	0.812	0.793	0.765
soybean-small	1.000	1.000	1.000	0.982	1.000
splice	0.837	0.809	0.767	0.951	0.920
threenorm	0.820	0.830	0.833	0.694	0.846
tic-tac-toe	0.860	0.872	0.976	0.980	0.983
twonorm	0.920	0.947	0.953	0.823	0.963
vehicle	0.714	0.721	0.695	0.693	0.741
vote	0.943	0.949	0.926	0.952	0.954
vote1	0.910	0.903	0.903	0.908	0.924
vowel	0.792	0.939	0.917	0.710	0.688
waveform	0.821	0.827	0.810	0.813	0.870
yeast	0.592	0.605	0.563	0.587	0.590
zip	0.899	0.868	0.961	0.894	0.946
mean	0.811	0.820	0.829	0.832	0.831
std	0.152	0.155	0.163	0.157	0.170

TABLE C.4

Accuracy results for ensembles with bagging using voting as the fusion function.